Name: Satyam Rai

UID: 2019130051

Subject: AI/ML

Experiment No.: 1

Aim: Implement Wumpus World problem.

Theory:

The Wumpus world is a simple world example to illustrate the worth of a knowledge-based agent and to represent knowledge representation. It was inspired by a video game **Hunt the Wumpus** by Gregory Yob in 1973.

The Wumpus world is a cave which has 4/4 rooms connected with passageways. So, there are total 16 rooms which are connected with each other. We have a knowledge-based agent who will go forward in this world. The cave has a room with a beast which is called Wumpus, who eats anyone who enters the room. The Wumpus can be shot by the agent, but the agent has a single arrow. In the Wumpus world, there are some Pits rooms which are bottomless, and if agent falls in Pits, then he will be stuck there forever. The exciting thing with this cave is that in one room there is a possibility of finding a heap of gold. So, the agent goal is to find the gold and climb out the cave without fallen into Pits or eaten by Wumpus. The agent will get a reward if he comes out with gold, and he will get a penalty if eaten by Wumpus or falls in the pit.

PEAS Description:

1. Performance measure:

(a) +1000 points for picking up the gold — this is the goal of the agent

(b) −1000 points for dying = entering a square containing a pit or a live Wumpus Monster

(c) −1 point for each action taken, and

(d) −10 points for using the arrow trying to kill the Wumpus -- so that the agent should avoid performing unnecessary actions.

2. Environment: A 4 × 4 grid of squares with. . .

(a) the agent starting from square [1, 1] facing right

(b) the gold in one square

(c) the initially live Wumpus in one square, from which it never moves

(d) maybe pits in some squares. The starting square [1, 1] has no Wumpus, no pit, and no gold – so the agent neither dies nor succeeds straight away.

3. Actuators: The agent can turn 90° left or right walk one square forward in the current direction, grab an object in this square, shoot the single arrow in the current direction, which flies in a straight line until it hits a wall or the Wumpus.

4. Sensors: The agent has 5 true/false sensors which report a stench when the Wumpus is in an adjacent square — directly, not diagonally breeze when an adjacent square has a pit glitter, when the agent perceives the glitter of the gold in the current square bump, when the agent walks into an enclosing wall (and then the action had no effect) scream, when the arrow hits the Wumpus, killing it

Code:

```
def setdir(i, j):
   si = i
   sj = j
   r = j + 1
   l = j - 1
   u = i - 1
   d = i + 1
   if i == 0 and j == 0:
      dirs[0] = (si, r)
      dirs[1] = ""
      dirs[2] = ""
```

```
            dirs[3] = (d, sj)
        elif i == 0 and j == 3:
            dirs[0] = ""
            dirs[1] = (si, l)
            dirs[2] = ""
            dirs[3] = (d, sj)
        elif i == 3 and j == 0:
            dirs[0] = (si, r)
            dirs[1] = ""
            dirs[2] = (u, sj)
            dirs[3] = ""
        elif i == 3 and j == 3:
            dirs[0] = ""
            dirs[1] = (si, l)
            dirs[2] = (u, sj)
            dirs[3] = ""
        elif i == 0:
            dirs[0] = (si, r)
            dirs[1] = (si, l)
            dirs[2] = ""
            dirs[3] = (d, sj)
        elif i == 3:
            dirs[0] = (si, r)
            dirs[1] = (si, l)
            dirs[2] = (u, sj)
            dirs[3] = ""
        elif j == 0:
            dirs[0] = (si, r)
            dirs[1] = ""
            dirs[2] = (u, sj)
            dirs[3] = (d, sj)
        elif j == 3:
            dirs[0] = ""
            dirs[1] = (si, l)
            dirs[2] = (u, sj)
            dirs[3] = (d, sj)
        else:
            dirs[0] = (si, r)
            dirs[1] = (si, l)
            dirs[2] = (u, sj)
            dirs[3] = (d, sj)


def allowed(ls):
    for objs in dirs:
        if objs != "":
            ls.append(objs)
    return ls


def checkwumpus(steps):
    if w[int(steps[0])][int(steps[1])] == 1:
        print("Player got killed")
        print(f'visited: {visited}')
        print('\n')
        for i in k:
            print(i)
        print('\n')
        exit(0)
    elif w[int(steps[0])][int(steps[1])] == 2:
        return 2
```

```
    else:
        return 0


def checkpit(steps):
    if p[int(steps[0])][int(steps[1])] == 3:
        print("Player got killed")
        print('\n')
        for i in k:
            print(i)
        print('\n')
        exit(0)
    elif p[int(steps[0])][int(steps[1])] == 4:
        return 4
    else:
        return 0


def stepintocell(step):
    ws = checkwumpus(step)
    pb = checkpit(step)
    return (ws, pb)


def action():
    global tl2, present, tv
    for step in tl:
        print(f'\nFor {step}:')
        if step in visited:
            print(step, "is visited\n")
            if int(step[0]) == pm and int(step[0]) == pn:
                print("1")
            elif '2' in str(k[int(step[0])][int(step[1])]):
                print("2")
        else:
            setdir(int(step[0]), int(step[1]))
            tl2 = tl2[0:0]
            tl2 = allowed(tl2)
            print("Connections of", step, ': ', tl2)
            for st in tl2:
                if st == present:
                    print(st, "is Present Cell")
                else:
                    print("checking ", st)
                    if st in start:
                        print(st, "is Start state")
                    elif st == present:
                        print(st, "is Present state")
                    elif st in visited:
                        print(st, "is visited")
                        if (k[int(st[0])][int(st[1])] in [(0, 4), (4, 0), (0, 2),
(2, 0)]) and (
                                k[int(present[0])][int(present[1])] in [(0, 2), (2,
0), (0, 4), (4, 0)]):
                            k[int(step[0])][int(step[1])] = 'S'
                            print(f'Putting Safe State at ({step[0]}, {step[1]})')
                            print("Visited :", visited)
                            return step
                    print("\n")
    print("______\n")
    return 0
```

```
def nextstep(ws, wp):
   global tl, present, prev, tmp, h
   print("\nPresent: ", present, end=' ')
   print("Previous: ", prev)
   setdir(ws, wp)
   tl = tl[0:0]
   tl = allowed(tl)
   if ws == gm and wp == gn:
      h = 1
      if h == 1:
         print("\nHurray!!Got the Gold", end=' ')
      print("\nPresently in %s returning back to %s" % (present, start))
      for steps in range(len(mypath), 0, -1):
         prev = present
         present = mypath[steps - 1]
      print("Now reached to :", present)
      exit(0)
   elif k[ws][wp] == "S":
      print(f'\n{present} is a Safe Cell')
      print("Allowed Steps", tl)
      for step in tl:
         tmp = step
         if step not in visited:
            print(step, "Not visited")
            prev = present
            print("Previous: ", prev)
            present = (step[0], step[1])
            print("Present: ", present)
            k[int(step[0])][int(step[1])] = stepintocell(step)
            print(f'\nStepping into ({step[0]}, {step[1]})')
            mypath.append(step)
            visited.append(step)
            print(f'visited: {visited}')
            print('\n')
            for i in k:
               print(i)
            print('\n')
            nextstep(int(step[0]), int(step[1]))
         else:
            print(step, "is already visited")
   else:
      print(f'\n{present} is not a Safe Cell', end=' ')
      print("\nConnections of ", present, ":", tl, end=' ')
      l = action()
      if l == 0:
         print("Stepping Back to", prev, "\n")
         mypath.pop()
         present = prev
         nextstep(int(prev[0]), int(prev[1]))
      else:
         print("Reached ", l)
         visited.append(l)
         print(f'visited: {visited}')
         prev = tmp
         present = l
         print("\nfrom here Stepping into ", l, end=' ')
         mypath.append(l)
         nextstep(int(l[0]), int(l[1]))
```

```
dirs = ["right", "left", "up", "down"]
gm = 1
gn = 1
pm = 3
pn = 0
h = 0
visited = []
w = [[2, 0, 0, 0], [1, 2, 0, 2], [2, 0, 2, 1], [0, 0, 0, 2]]
print('wumpus:')
for i in w:
   print(i)
p = [[0, 0, 4, 3], [0, 4, 3, 4], [0, 0, 4, 0], [0, 4, 3, 4]]
print('pit:')
for i in p:
   print(i)
k = [["", "", "", ""], ["", "", "", ""], ["", "", "", ""], ["", "", "",
""]]
k[pm][pn] = 'S'
print('\n')
for i in k:
   print(i)
print('\n')
tl = []
tl2 = []
tmp = ""
tv = 1
mypath = []
prm = pm
prn = pn
start = (pm, pn)
visited.append(start)
print(f'visited: {visited}')
print(f'Starting from ({pm}, {pn})')
prev = (pm, pn)
present = (pm, pn)
mypath.append(start)
nextstep(pm, pn)
print(present)
print('\n')
for i in k:
   print(i)
print('\n')
```

Output:

```
C:\Users\soham\anaconda3\python.exe C:/Users/soham/Downloads/wumpus_soham.py
wumpus:
[2, 0, 0, 0]
[1, 2, 0, 2]
[2, 0, 2, 1]
[0, 0, 0, 2]
pit:
[0, 0, 4, 3]
[0, 4, 3, 4]
[0, 0, 4, 0]
[0, 4, 3, 4]


['', '', '', '']
['', '', '', '']
['', '', '', '']
['S', '', '', '']


visited: [(3, 0)]
Starting from (3, 0)

Present:  (3, 0) Previous:  (3, 0)

(3, 0) is a Safe Cell
Allowed Steps [(3, 1), (2, 0)]
```

```
(3, 1) Not visited
Previous:  (3, 0)
Present:  (3, 1)

Stepping into (3, 1)
visited: [(3, 0), (3, 1)]


['', '', '', '']
['', '', '', '']
['', '', '', '']
['S', (0, 4), '', '']



Present:  (3, 1) Previous:  (3, 0)

(3, 1) is not a Safe Cell
Connections of  (3, 1) : [(3, 2), (3, 0), (2, 1)]
For (3, 2):
Connections of (3, 2) :  [(3, 3), (3, 1), (2, 2)]
checking  (3, 3)
(3, 1) is Present Cell
checking  (2, 2)

For (3, 0):
```

```
(3, 0) is visited


For (2, 1):
Connections of (2, 1) :  [(2, 2), (2, 0), (1, 1), (3, 1)]
checking  (2, 2)
checking  (2, 0)
checking  (1, 1)
(3, 1) is Present Cell
------

Stepping Back to (3, 0)


Present:  (3, 0) Previous:  (3, 0)

(3, 0) is a Safe Cell
Allowed Steps [(3, 1), (2, 0)]
(3, 1) is already visited
(2, 0) Not visited
Previous:  (3, 0)
Present:  (2, 0)

Stepping into (2, 0)
visited: [(3, 0), (3, 1), (2, 0)]
```

```
['', '', '', '']
['', '', '', '']
[(2, 0), '', '', '']
['S', (0, 4), '', '']



Present:  (2, 0) Previous:  (3, 0)

(2, 0) is not a Safe Cell
Connections of  (2, 0) : [(2, 1), (1, 0), (3, 0)]
For (2, 1):
Connections of (2, 1) :  [(2, 2), (2, 0), (1, 1), (3, 1)]
checking  (2, 2)
(2, 0) is Present Cell
checking  (1, 1)
checking  (3, 1)
(3, 1) is visited
Putting Safe State at (2, 1)
Visited : [(3, 0), (3, 1), (2, 0)]
Reached  (2, 1)
visited: [(3, 0), (3, 1), (2, 0), (2, 1)]

from here Stepping into  (2, 1)
```

```
from here Stepping into  (2, 1)
Present:  (2, 1) Previous:  (2, 0)

(2, 1) is a Safe Cell
Allowed Steps [(2, 2), (2, 0), (1, 1), (3, 1)]
(2, 2) Not visited
Previous:  (2, 1)
Present:  (2, 2)

Stepping into (2, 2)
visited: [(3, 0), (3, 1), (2, 0), (2, 1), (2, 2)]



['', '', '', '']
['', '', '', '']
[(2, 0), 'S', (2, 4), '']
['S', (0, 4), '', '']



Present:  (2, 2) Previous:  (2, 1)

(2, 2) is not a Safe Cell
Connections of  (2, 2) : [(2, 3), (2, 1), (1, 2), (3, 2)]
For (2, 3):
```

```
Connections of (2, 3) :  [(2, 2), (1, 3), (3, 3)]
(2, 2) is Present Cell
checking  (1, 3)
checking  (3, 3)

For (2, 1):
(2, 1) is visited


For (1, 2):
Connections of (1, 2) :  [(1, 3), (1, 1), (0, 2), (2, 2)]
checking  (1, 3)
checking  (1, 1)
checking  (0, 2)
(2, 2) is Present Cell

For (3, 2):
Connections of (3, 2) :  [(3, 3), (3, 1), (2, 2)]
checking  (3, 3)
checking  (3, 1)
(3, 1) is visited


(2, 2) is Present Cell
------
```

```
Stepping Back to (2, 1)


Present:  (2, 1) Previous:  (2, 1)

(2, 1) is a Safe Cell
Allowed Steps [(2, 2), (2, 0), (1, 1), (3, 1)]
(2, 2) is already visited
(2, 0) is already visited
(1, 1) Not visited
Previous:  (2, 1)
Present:  (1, 1)

Stepping into (1, 1)
visited: [(3, 0), (3, 1), (2, 0), (2, 1), (2, 2), (1, 1)]


['', '', '', '']
['', (2, 4), '', '']
[(2, 0), 'S', (2, 4), '']
['S', (0, 4), '', '']



Present:  (1, 1) Previous:  (2, 1)
```

```
Hurray!!Got the Gold
Presently in (1, 1) returning back to (3, 0)
Now reached to : (3, 0)

Process finished with exit code 0
```

Conclusion: In this experiment I learnt about the Wumpus World problem in Artificial Intelligence. I also learnt how to implement it in Python.